Universidade Federal do Rio de Janeiro

Núcleo de Computação Eletrônica

Hélio Ricardo Souza de Lemos

ALTA DISPONIBILIDADE NA WEB:

Web Clustering

Rio de Janeiro

2008

Hélio Ricardo Souza de Lemos

ALTA DISPONIBILIDADE NA WEB:

Web Clustering

Monografia apresentada para obtenção do título de Especialista em Gerência de Redes de Computadores no Curso de Pós-Graduação Lato Sensu em Gerência de Redes de Computadores e Tecnologia Internet do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro – NCE/UFRJ.

Orientador:

Moacyr Henrique Cruz de Azevedo, M.Sc., COPPE/UFRJ

Rio de Janeiro

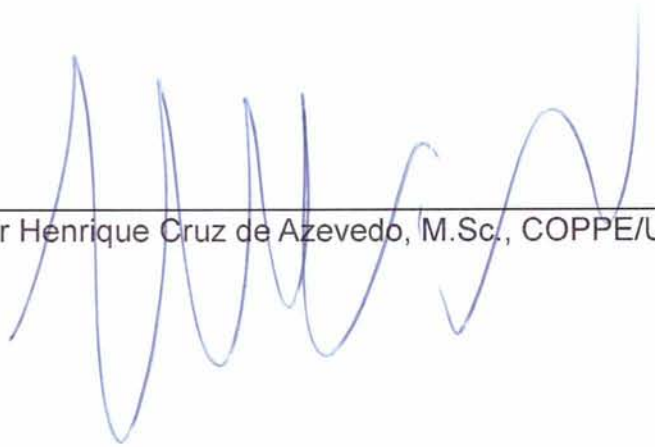
2008

Hélio Ricardo Souza de Lemos

ALTA DISPONIBILIDADE NA WEB: Web Clustering

Monografia apresentada para obtenção do título de Especialista em Gerência de Redes de Computadores no Curso de Pós-Graduação Lato Sensu em Gerência de Redes de Computadores e Tecnologia Internet do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro – NCE/UFRJ.

Aprovada em outubro de 2008.



Moacyr Henrique Cruz de Azevedo, M.Sc., COPPE/UFRJ

Dedico a todos que colaboram de alguma forma à disseminação da cultura Open Source.

AGRADECIMENTOS

Gostaria de agradecer aos companheiros de classe, docentes e funcionários do NCE, pela motivação e dedicação a mim empregada.

RESUMO

LEMOS, Hélio Ricardo Souza de. **ALTA DISPONIBILIDADE NA WEB: Web Clustering**. Monografia (Especialização em Gerência de Redes e Tecnologia Internet). Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2008.

No paradigma dos sistemas modernos voltados para web, surge o crescimento da necessidade da contínua disponibilidade de serviços online para os usuários. Demoras ou interrupções nas interações cliente-servidor podem custar caro para os provedores do serviço e para os próprios clientes. Soluções de Alta Disponibilidade são utilizadas para tornar os serviços disponíveis, mesmo com falhas críticas em sua infra-estrutura. Em ambientes altamente disponíveis uma falha é automaticamente detectada e a carga de criticidade gerada é diluída dentro da estrutura . Este trabalho apresenta um modelo de alta disponibilidade para sistemas web.

ABSTRACT

LEMOS, Hélio Ricardo Souza de. **ALTA DISPONIBILIDADE NA WEB: Web Clustering**. Monografia (Especialização em Gerência de Redes e Tecnologia Internet). Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2008.

In the modern web-based systems paradigm, there is an increasing need for continuous online services availability to the users. Delays or interruptions in client-server interactions may be costly equally to the service providers and the customers. High availability solutions are used to withstand the service availability due to infrastructure critical failures. In high availability environments a failure is automatically detected and then the critical load generated is diluted into the structure. This paper presents a high availability model for web systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

B2B	Business to Business
B2C	Business to Consumer
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Protocol
TCP	Transmission Control Protocol
CGI	Common Gateway Interface
UDP	User Datagram Protocol
IP	Internet Protocol
LVS	Linux Virtual Server
VIP	Virtual Internet Protocol
NAT	Network Address Translation
ISP	Internet Service Provider
VRRP	Virtual Router Redundancy Protocol
MAC	Media Access Protocol
CARP	Commom Address Redundancy Protocol
UCARP	Userland Common Address Redundancy Protocol
SSH	Secure Shell
LAN	Local Area Network
NFS	Network File System
I/O	Input/Output
SNMP	Simple Network Management Protocol
MD5	Message Digest Algorithm 5

SUMÁRIO

1 **INTRODUÇÃO**

Desde seu início, a Internet tem crescido de forma exponencial. Aumentam o número de internautas, a infra-estrutura, o volume de conteúdo e o avanço tecnológico dos recursos de rede, com diminuição de latência e aumento do throughput. Em junho de 2007 o número de usuários da grande rede beirava 1.2 bilhões, aproximadamente 17% da população mundial.

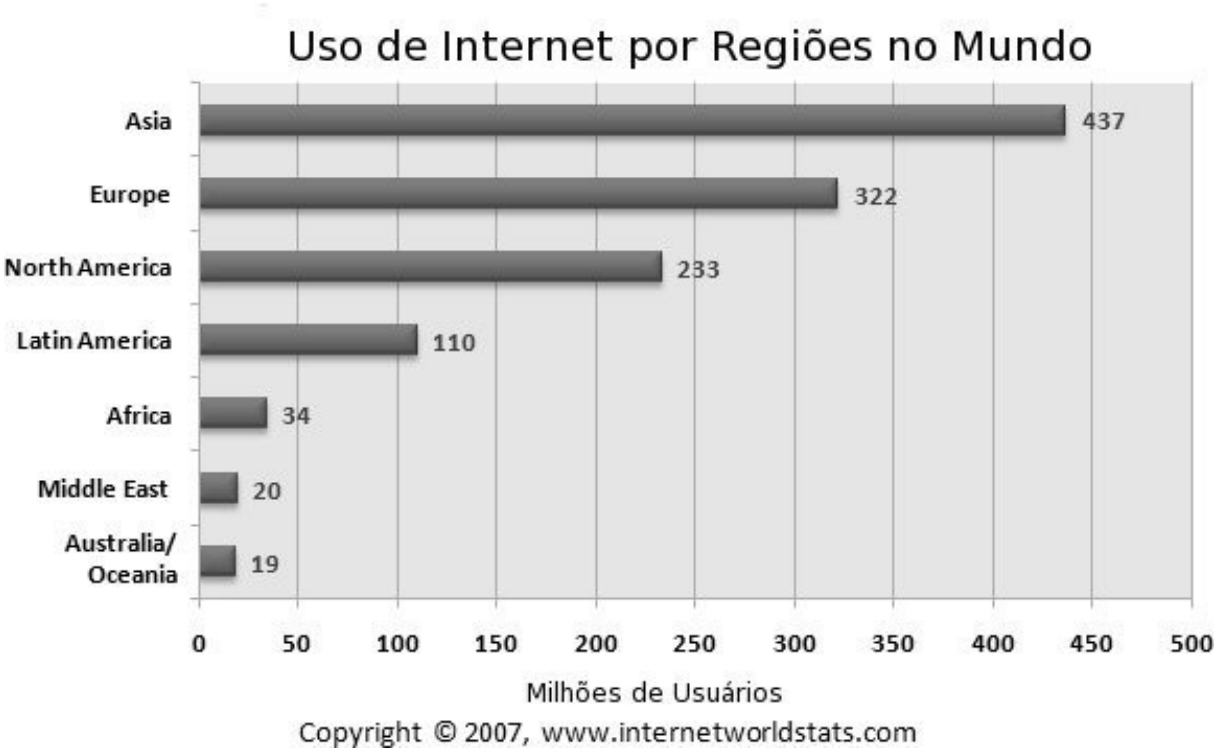


Figura 1 - Uso de Internet dividido por regiões do Mundo

Essa explosão de tráfego impulsionou o aumento de sites oferecendo e-commerce, comunidades e outros recursos com alta carga de acesso. Tudo isto face ao desafio de mantê-los sempre disponíveis e tolerante à falhas.

Com o aquecimento das áreas de B2B e B2C nos últimos anos, é trivial deduzir que aumenta a preocupação das empresas no que diz respeito à disponibilidade de seus serviços. Com a demanda sendo online, mesmo pequenos downtimes podem impactar economicamente estas instituições.

Google

	Full Year					2006			
Revenue	2002	2003	2004	2005	2006	Q1	Q2	Q3	Q4
Revenues	439,508	1,465,934	3,189,223	6,138,560	10,604,917	2,253,755	2,455,991	2,689,673	3,205,498
Y/Y Growth rate	*409%*	*234%*	*118%*	*92%*	*73%*	*79%*	*77%*	*70%*	*67%*
Q/Q Growth rate	*NA*	*NA*	*NA*	*NA*	*NA*	*17%*	*9%*	*10%*	*19%*

Figura 2 - Rendimentos do ano de 2006 do Google *(fonte: http://www.google.com/finance?q=GOOG)*

Em 2006 o total de rendimentos do Google foi pouco mais de $10,5 bilhões (http://investor.google.com/fin_data.html). Supondo uma distribuição uniforme de tráfego durante o ano, estes dados traduzem um rendimento diário de $29 milhões ou $335 por segundo (!).

Uma disponibilidade de 99% é considerada, neste caso, equivocadamente próxima da perfeição. O 1% restante custaria para o Google mais de $100 milhões de dólares por ano. Esta indisponibilidade levaria, além da perda financeira direta, à falta de credibilidade da empresa.

Tabela 1 (Parte 1) - Downtimes do Google 09/2007 *(fonte: Serviço online de monitoramento de uptime Pingdom)*

Medida de Downtime entre 1 Setembro de 2006 até 1 Setembro de 2007			
Country	**URL**	**Downtime in minutes**	**Uptime over a year**
Brazil	www.google.com.br	3	99,999%
Netherlands	www.google.nl	11	99,998%
India	www.google.co.in	12	99,998%
Thailand	www.google.co.th	13	99,997%
Japan	www.google.co.jp	15	99,997%
Canada	www.google.ca	16	99,997%
Mexico	www.google.com.mx	16	99,997%
Egypt	www.google.com.eg	16	99,997%
Chile	www.google.cl	17	99,997%

Tabela 1 (Parte 2) - Downtimes do Google 09/2007 *(fonte: Serviço online de monitoramento de uptime Pingdom)*

France	www.google.fr	19	99,996%
Greece	www.google.gr	19	99,996%
United Arab Emirates	www.google.ae	20	99,996%
United Kingdom	www.google.co.uk	20	99,996%
Poland	www.google.pl	20	99,996%
Argentina	www.google.com.ar	21	99,996%
Hong Kong	www.google.com.hk	22	99,996%
Spain	www.google.es	22	99,996%
Italy	www.google.it	22	99,996%
Belgium	www.google.be	22	99,996%
Switzerland	www.google.ch	22	99,996%
Australia	www.google.com.au	26	99,995%
Romania	www.google.ro	27	99,995%
Saudi Arabia	www.google.com.sa	27	99,995%
Malaysia	www.google.com.my	28	99,995%
Germany	www.google.de	29	99,994%
United States	www.google.com	31	99,994%

Este estudo visa apresentar conceitos, técnicas e ferramentas para a implantação de ambientes web com o máximo de disponibilidade possível.

2 **FUNDAMENTAÇÃO TEÓRICA**

2.1 ALTA DISPONIBILIDADE

Para entendermos as implicações da alta disponibilidade em sistemas computacionais, precisamos definir alguns termos como: disponibilidade, alta disponibilidade e alta disponibilidade computacional.

2.1.1 **Disponibilidade**

O termo **disponível** descreve um sistema que provê um nível de serviço específico tanto quanto necessário. Na computação a disponibilidade é normalmente entendida pelo período de tempo que um serviço está disponível (ex: 24x7) ou o tempo necessário para um sistema responder seus usuários (ex: 1 requisição por segundo). Qualquer perda de serviço, planejada ou não, é conhecido como um **outage**. E a duração de um outage, em alguma unidade de tempo, é chamada de **downtime**.

2.1.2 **Alta Disponibilidade**

Um sistema **altamente disponível** é caracterizado por ser projetado para evitar a perda de serviço através da redução e gerenciamento de falhas, assim como minimizar os downtimes planejados. Temos como exemplo de serviços altamente disponíveis a nossa rede elétrica e telefonia fixa. E mesmo assim, temos problemas de indisponibilidade.

2.1.3 **Alta Disponibilidade Computacional**

A **alta disponibilidade computacional** define-se como sistemas de computadores que são projetados e gerenciados para operar com o mínimo de **downtimes** planejados ou não. A alta disponibilidade não é absoluta. As diferentes necessidades em cada tipo de negócio, fazem-na tornar-se diversificada. Companhias internacionais e muitos sites de Internet requerem acessos quase instantâneos à sua base de dados. Instituições financeiras precisam estar

disponíveis para transferência de fundos dia e noite, sete dias por semana. Em outras palavras, alguns tipos de negócios necessitam de dezoito horas por dia de disponibilidade, mas durante este período o tempo de resposta para o processamento de uma transação precisa estar na casa dos milisegundos.

2.2 WEB SERVERS

2.2.1 **O que são**

Web server é um programa de computador responsável por aceitar pedidos HTTP de clientes e serví-los com repostas HTTP, incluindo opcionalmente dados como texto e imagens. Normalmente, os pedidos referem-se à páginas html e são feitos através de browsers. O processo se inicia com o usuário digitando um endereço web com o seguinte formato: **protocolo://servidor/pedido-URI**

O browser começa abrindo uma conexão TCP com o servidor (SYN). No passo seguinte o navegador manda o pedido para o servidor (DAT). Ele o faz enviando uma mensagem no seguinte formato: **GET [diretório(URI)] [HTTP/versão]**

O servidor localiza o documento e manda a seguinte resposta HTTP (DAT):

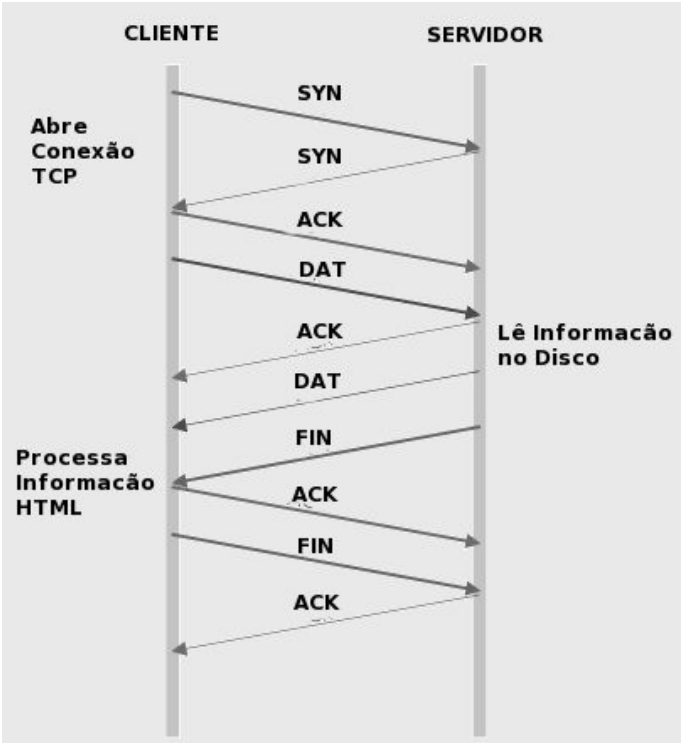


Figura 3 - Fluxo HTTP

HTTP/[ver] [código] [texto]
Campo1: valor1
Campo2: valor2

... conteúdo do documento....

Explicando os termos acima: *ver* é a versão do HTTP, código é um número de 3 algarismos, geralmente 200 para dizer que está tudo OK, e depois um texto que traduz o significado deste número para uma linguagem conhecida (geralmente é o próprio "ok"). Seguem-se algumas informações usadas pelo cabeçalho, como data, tamanho do arquivo etc. Depois de uma linha em branco, vem a informação do documento propriamente.

Quadro 1 – Exemplo de cabeçalho HTTP

```
HTTP/1.0 200 OK
Server: Netscape-Communications/1.1
Date: Tuesday, 25-Nov-99 01:22:03 GMT
Last-modified: Thursday, 20-Nov-99 10:44:33 GMT
Content-length: 6372
Content-type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
... conteúdo do documento ...
```

Os campos exemplificados são de fácil compreensão. Chamamos a atenção para o chamado "content-type", que tem como valor "text/html". Este campo é o que permite ao navegador saber que tipo de documento está sendo trafegado, e, dependendo do tipo, abri-lo na própria janela ou chamar um outro programa para fazê-lo. Outros exemplos de tipos de documento são: "text/plain" (texto puro), "image/gif" (imagem do tipo gif), "image/jpg" (imagem do tipo jpg) entre outros.

Alguns documentos são abertos dentro de próprio documento html, como é o caso das imagens. Outros são visualizados também dentro do documento html, mas requerem um plug-in, como é o caso das animações "flash" (da Macromedia).

Um importante conceito aqui é que para o navegador não importa como o servidor produz a informação que lhe manda. Portanto, esta pode ser estática, dinâmica, gerada por um programa (cgi), processada de mil e uma maneiras, que

ele não vai saber: só o que importa é saber o tipo de documento para que ele possa usar o programa ou plug-in necessário para abrí-lo.

Como na web não é possível prever a que hora se dará essa conexão, os web servers precisam estar disponíveis dia e noite. Genericamente tudo o que se enquadre no conceito de arquivo pode ser enviado como resultado de um pedido HTTP.

2.2.2 **HTTP Servers Estáticos**

Servidores estáticos servem seus arquivos de conteúdos à partir de seu sistema de arquivos.

Fuxo pedido/resposta gerado quando se acessa uma página estática:

Quadro 2 - Conteudo do arquivo teste.html

```
<html>
 <body>
  <p>olá mundo</p>
 </body>
</html>
```

Quadro 3 - Pedido HTTP Estático

```
GET /teste.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-pdf */*
Accept-Language: en-gb,pt;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)
Host: localhost:79
Connection: Keep-Alive
Cookie: infoview_userCultureKey=useBrowserLocale
```

Quadro 4 (Parte 1) - Resposta HTTP Estática

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET
Date: Thu, 25 May 2006 14:02:51 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Thu, 25 May 2006 14:02:12 GMT
```

Quadro 4 (Parte 2) - Resposta HTTP Estática

```
ETag: "cd3bdd2380c61:ba9"
Content-Length: 54

<html>
 <body>
  <p>olá mundo</p>
 </body>
</html>
```

Embora estes valores variem de acordo com o browser que utilizarmos e com o servidor web que responde a este pedido HTTP, muito do conteúdo será sempre igual.

2.2.3 **Application Servers**

Servidores de aplicação oferecem seus conteúdos dinamicamente. O pedido, depois de recebido, é processado pelo servidor web que vai criar dinamicamente o conteúdo que depois será enviado para o cliente.

As páginas dinâmicas têm a vantagem de poderem ser programadas, ou seja, usam alguma linguagem de programação. Podemos criar programas que rodam no servidor web, eventualmente acessando um banco de dados cujo resultado é enviado para o browser.

Fluxo pedido/resposta gerado quando se acessa uma página dinâmica

Quadro 5 - Conteudo do arquivo teste.asp

```
<html>
 <body>
  <%
  for i=1 to 10
        Response.Write("<p>olá mundo</p>")
  next
  %>
 </body>
</html>
```

Se o browser tentar ter acesso a este arquivo, a sequência pedido / resposta iria produzir os seguintes comandos

Quadro 6 - Pedido HTTP Dinâmico

```
GET /teste.asp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-
shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*
Accept-Language: en-gb,pt;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
1.1.4322; .NET CLR 2.0.50727)
Host: localhost:79
Connection: Keep-Alive
Cookie: infoview_userCultureKey=useBrowserLocale;
ASPSESSIONIDQSRCCSAS=KJLFNNNCNHKODOIOCIICJFBA
```

Quadro 7 - Resposta HTTP Dinâmica

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Thu, 25 May 2006 14:20:34 GMT
X-Powered-By: ASP.NET
Content-Length: 198
Content-Type: text/html
Cache-control: private
<html>
 <body>
  <p>olá mundo</p><p>olá mundo</p><p>olá mundo</p><p>olá mundo</p><p>olá
mundo</p><p>olá mundo</p><p>olá mundo</p><p>olá mundo</p><p>olá
mundo</p><p>olá mundo</p>
 </body>
</html>
```

Tanto o pedido/resposta de um arquivo estático como o de um arquivo dinâmico geram fluxos de informação praticamente iguais, isto é, na web a informação que circula é essencialmente a mesma. A diferença é que um arquivo dinãmico tem que ser primeiro processado pelo servidor web.

3 **WEB CLUSTERING**

3.1 LOAD BALANCING CLUSTERS

Clustering é a integração entre software e hardware que possibilita um pool de servidores trabalharem como um único sistema, apresentando uma entidade única para os clientes.

Load Balancing Clusters oferece o mais prático sistema que o mercado web necessita, sendo implementado na camada 4. Sua função é distribuir as requisições para os servidores na camada de transporte, como TCP ou UDP, distribuindo a carga de processamento entre os nós que oferecem o serviço. O load balancer, ou balanceador de carga, distribui as conexões de rede vindas de clientes que conhecem um único endereço IP de um serviço, para um pool de servidores. Uma vez estebelecida a conexão entre o cliente e o servidor num transporte orientado a conexão, o load balancer escolhe um servidor, normalmente sem analisar o conteúdo da requisição.

Diversos são os benefícios destes métodos, dentre eles:

- *Escaláveis*: Sistemas em Clustering são escaláveis devido a capacidade de aumentar sua performance através do acréscimo de um ou múltiplos nós ao Cluster. Essa é uma grande vantagem em sistemas que a carga cresceu mais do que o esperado e uma simples adição de hardware ao Cluster aumenta sua capacidade.
- *Disponíveis*: Caso haja algum problema em parte dos nós do cluster, o serviço continua operante. A redundância é uma das consequências mais importantes, que é ocasionada ao implementar estas técnicas. Com um bom planejamento, elimina-se a possibilidade de um resultado catastrófico.
- *Gerenciáveis*: Seu gerenciamento é bastante simples, pois existe um número imenso de ferramentas para este fim no mercado. Estas oferecem a inspeção do status do cluster, tornam manual ou automático o balanceamento de carga, de fácil parametrização e upgrades sem trauma.
- *Redução de Custos*: Devido a redução de downtimes oferecido pelo Load Balancing Cluster, pode-se reduzir os custos de suporte e de dinheiro perdido com a saída do ar do serviços.

3.2 LVS

3.2.1 **Definição**

Linux Virtual Server é um cluster de servidores que aparentam ser um só para clientes externos. Este único servidor é chamado de "virtual server". Os servidores reias são os real servers que ficam sob o controle do director (balanceador de carga).

Por exemplo, quando uma nova conexão é requisitada de um cliente para um serviço oferecido via LVS, HTTP, o director vai escolher um real server para o cliente. Então todos os pacotes vindos do cliente são repassados atrevés do director para um real server específico. Esta associação se manterá enquanto durar a vida da conexão TCP (ou UDP). Na próxima conexão TCP, o director escolherá um novo real server, que pode ou não, ser o mesmo. Quando um web browser se conectar a um LVS servindo uma página web contendo diversos links (imagens, html), cada hit poderá ser em um real server diferente.

O director apresenta um IP chamado IP virtual (VIP) para os clientes. Este é o endereço do serviço que será balanceado.

A redundância é a principal função do LVS no quesito alta disponibilidade. Os servidores podem ser removidos do LVS por diversos motivos como atualização de sistemas ou falha de hardware, e prontamento colocados de volta sem nenhuma interrupção do serviço. Outro fator de extrema importância é a capacidade de adaptação. Caso seja esperado uma mudança do throughput, gradual ou rápida, o aumento ou diminuição do número de servidores poderá ser feito de forma transparente para os usuários.

Seu funcionamento é como de um switch de camada 4. A semântica padrão do cliente-servidor continua preservada. Cada cliente pensa que está diretamente conectado ao real server. Cada servidor pensa que está conectado diretamente ao cliente. Nem o cliente nem o servidor sabem que as conexões sofrem a intervenção do director. Um real server do LVS não coopera, ele não sabe sequer da existência de outros real servers.

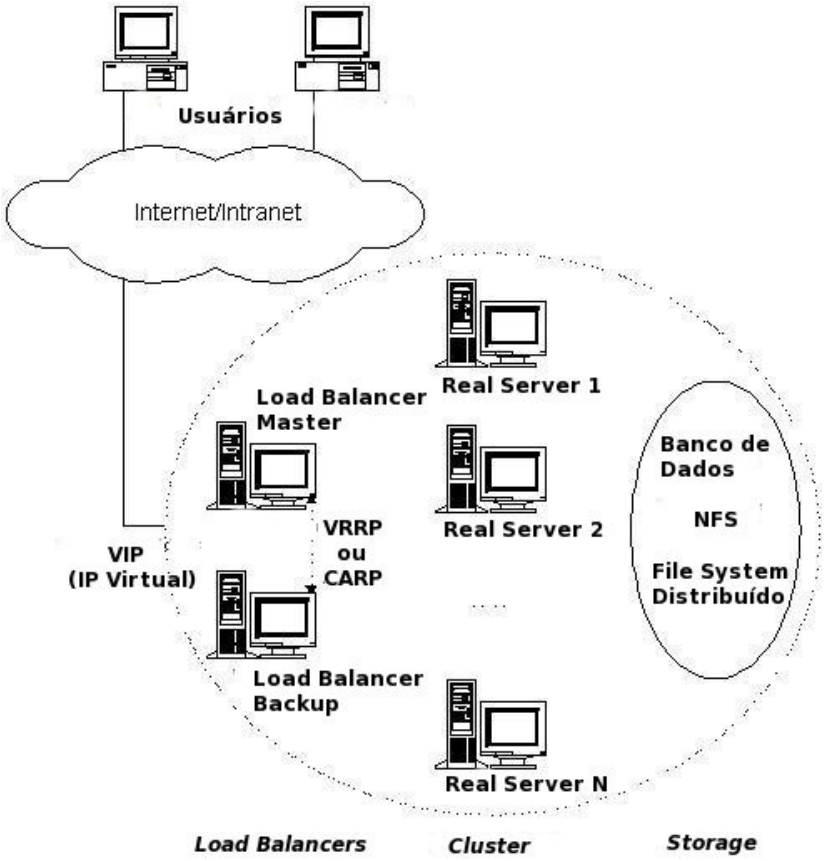


Figura 4 - LVS

3.2.2 **LVS-NAT**

É baseado no LocalDirector da Cisco. Foi o primeiro método de LVS a ser desenvolvido, onde o default gateway dos real servers é o director. Neste, o director, reescreve o destino dos pacotes entrantes, do IP virtual para o IP real, e então os direciona para os servidores reais. As respostas destes são enviadas para o director, onde são novamente reescritas e retornadas para o cliente com o endereço de origem modificado do IP real para o IP virtual.

O IP virtual é o único conhecido pelo cliente. Em um NAT normal, o mascaramento é feito nos pacotes originados atrás na NAT box. Com o LVS-NAT, os pacotes entrantes são reescritos. Esta ação é conhecida como desmascaramento.

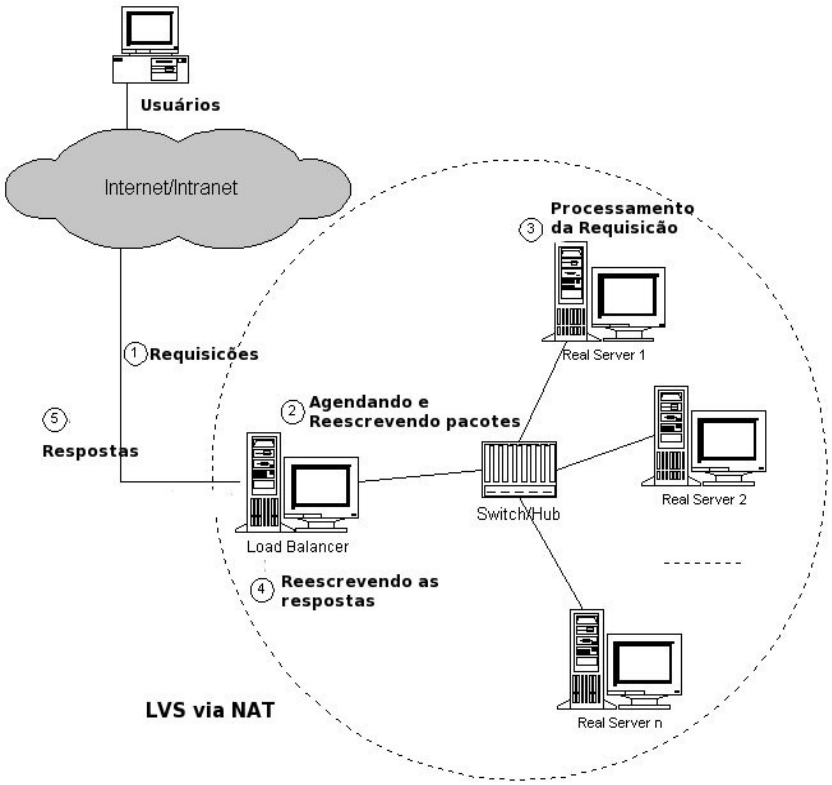


Figura 5 – LVS-NAT

3.2.3 **Persistência de Sessão**

Em um cluster LVS assume-se que cada conexão é independente de qualquer outra. Logo, cada conexão pode ser sinalizada à um real server independentemente de associações passadas, presentes e futuras. Porém, muitas vezes duas conexões de um mesmo cliente precisam ser sinalizadas para o mesmo real server por razões de performance e funcionalidade do serviço oferecido.

Se for utilizado algum tipo de autenticação, apenas o real server que fez o handshake de chaves sabe que o cliente está autenticado. Uma vez trocado de servidor, o cliente precisa recomeçar o processo.

Uma solução seria adicionar ao LVS uma persistência baseada em portas. Neste modelo, quando o primeiro cliente acessa o serviço, o director irá criar um template de conexão entre o cliente solicitante e o real server escolhido. Com esses dados será criada uma entrada para esta conexão em uma tabela. O tempo de vida do template e seu timeout são configuráveis e o template não expira até todas as conexões expirarem.

Um exemplo de template de uma conexão seria: <cip, 0, vip, 0, rip, 0>, onde o cip é o endereço IP do cliente, vip é o IP virtual e rip o IP do real server. O valor zero (0), significa que o acesso a qualquer porta dos três endereços será persistente. No caso da conexão HTTP tem-se: <cip, 0, vip, 80, rip, 80>, onde pode-se especializar qualquer serviço e trabalhar com diferentes portas nos real servers.

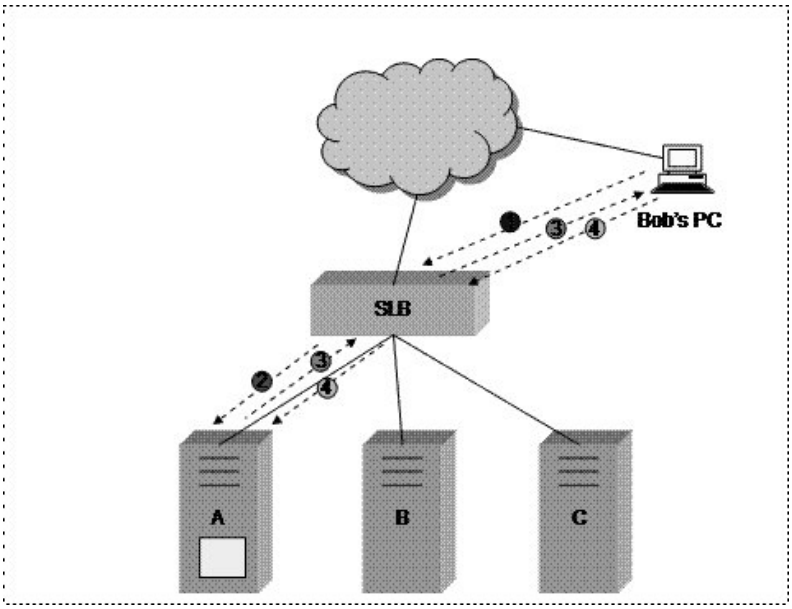


Figura 6 - Persistência de Sessão

3.3 FAILOVER

3.3.1 **Princípio**

Failover é a capacidade de trocar automaticamente para um servidor, sistema ou rede em modo redundante ou standby, em caso de falha ou término abrupto de um servidor, sistema ou rede previamente ativa. O Failover deve acontecer sem intervenção humana e, normalmente, sem nenhum alerta que necessite de alguma interação.

Até entao, vemos como único ponto de falha, neste conceito de Load Balancing Clusters, o próprio balanceador de carga (director). Se uma parte, ou a maioria, dos nós (real servers) cair, o serviço continuará no ar. Porém se o director falhar perderíamos o VIP, que é o único endereçõ que o cliente conhece. O serviço estaria fora do ar e com isso downtime, perda financeira, etc.

Em um sistema altamente disponível seria inaceitável tal possibilidade. Logo seria necessário a implementação de um protocolo de redundância, altamente confiável e performático, entre um director principal e um de backup. Este deverá contemplar o esquema de Fallback, que é o processo de restauração do estado de Failover para o estado original.

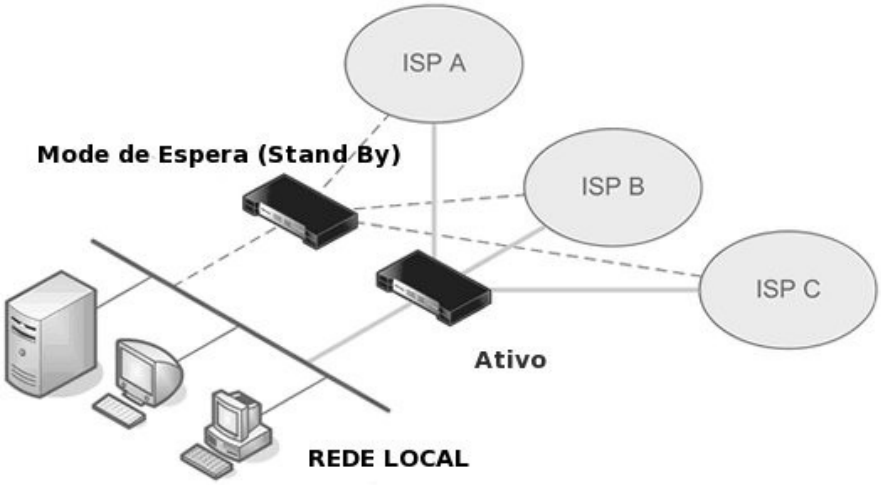


Figura 7 – Failover

3.3.2 **VRRP**

O VRRP especifica um protocolo de eleição, que dinamicamente sinaliza responsabilidade a algum servidor (virtual router) à tornar-se o VRRP router da LAN. O VRRP router que controla e envia pacotes para os endereços IP associados aos virtual routers é chamado de Master. As mensagens enviadas por ele são denominadas VRRP Advertisement e possuem uma prioridade que garante a vitória na eleição. Cada virtual router tem um único endereço MAC alocado para si. Este endereço é utilizado como origem em todas as mensagens VRRP enviadas pelo Master router. Todas as mensagens do protocolo são feitas usando datagramas IP multicast. Um virtual router é definido pelo virtual router identifier (VRID) e um set de endereços IP. Este pode associar um virtual router com diversos endereços reais em uma interface.

Para minimizar o tráfego de rede, apenas o Master envia as mensagens periódicas VRRP Advertisement com prioridade mais alta para cada virtual router. Um Backup router não tenta se eleger como Master sem uma prioridade mais alta do

que o VRRP Advertisement que está recebendo. Se um Master tornar-se indisponível, então o Backup com maior prioridade irá virar Master após um delay curtíssimo. Isto proporciona uma transição controlada de responsabilidade do virtual server, com o mínimo de interrupção do serviço. É o protocolo mais utilizado atualmente para Failover em ambientes web.

3.3.3 **CARP**

O Common Address Redundancy Protocol é um protocolo que possibilita múltiplos hosts em uma mesma rede local compartilhar um ou mais endereços IP. Seu objetivo principal é oferecer redundância Failover. Seu modo de funcionamento e eleição é bastante parecido com o VRRP, mas o CARP difere em alguns aspectos significativos. Ele foi desenhado para prover segurança e ser um protocolo independente, com isso criptografa todas as mensagens de Advertisement com SHA-1 HMAC e tem suporte full ao IPv4 e IPv6 (este, não suportado no VRRP).

Existe um port para Linux chamado Userland CARP (UCARP), que trata do mesmo protocolo, porém com um diferencial interessante. O Userland significa que ele funciona em modo usuário, o que facilita o processo de implementação por não precisar aplicar nenhum patch ao kernel.

Com isso torna-se uma opção muito segura para implementarmos uma redundância sólida, onde teremos dois, ou mais, load balancers (director), sendo um Master e outro em Backup pronto para assumir a posição principal, em caso de falha.

3.4 SINCRONISMO DE DADOS

3.4.1 **Objetivo**

Ao manter um web server, é simples a um programador ou designer fazer upload de arquivo para este servidor. Imaginemos duas máquinas, o trabalho seria dobrado. No caso de termos dezenas de servidores, o crescimento do trabalho cresceria proporcionalmente, sem contar no versionamento destes arquivos, onde a mesma estrutura deveria existir igualmente em todos. Não haveria escala na manutenção.

Uma solução seria o upload destes arquivos para um repositório único, e a partir deste, haveria um sincronismo de dados para todos os outros servidores.

Porém esbarraríamos no problema do tempo de sincronismo, onde haveria uma janela de desatualização de conteúdo entre os servidores. Uma premissa para o sincronismo ocorrer, é a necessidade do software que faça esta tarefa mantendo a integridade dos dados e sem comprometer a performance.

3.4.2 **RSync**

O RSync é um software open-source que sincroniza arquivos e diretórios de um local para outro, remoto ou não, com uma taxa de transferência minimizada por um algoritmo de encoding próprio.

Ele suporta compressão e encriptação através do protocolo SSH. Toda autenticação é feita através de contas comuns do sistema operacional. Todo processo pode ser agendado, de forma que uma estação central espelha todos os dados à múltiplos servidores.

A sincronização ocorreria de tempo em tempo, pré-definido, ou com interação humana. Com isso tem-se escala de manutenção e, naturalmente, um sistema de backup de dados distribuído.

Porém o RSync não resolve o problema da janela de desatualização.

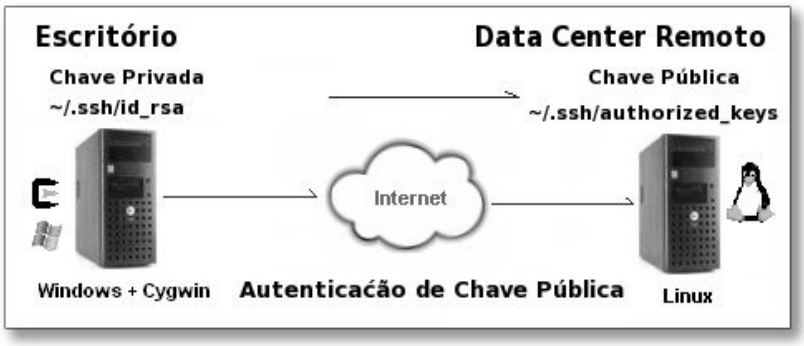


Figura 8 – Rsync

3.4.3 **NFS**

O protocolo Network File System foi desenvolvido para possibilitar que máquinas possam montar uma partição de disco de uma máquina remota, como se fosse uma partição de disco local. De forma síncrona, consegue-se compartilhar dados através da rede. Sua versão 3 assume uma implementação stateless, onde

não sao guardados os estados das conexões. Isto é uma vantagem sobre os outros, principalmente quando acontece um crash de sistema. Em modo stateless o cliente apenas fica tentando conectar no servidor até o mesmo voltar. O cliente não precisa saber que o servidor está "crasheado".

Todas as características remotas são apresentadas localmente. Com isso, torna-se um grande repositório de dados onde todos os servidores que oferecem conteúdo terão acesso de forma síncrona e com modificações em tempo real. Como estamos dentro de uma LAN, onde a velocidade e confiabilidade da rede são bastante controlados, implementar um servidor de dados que serve todo o conteúdo estático, via NFS, aos Web Servers, seria bastante funcional.

Em uma empresa de internet que possui bastante servidores (real servers), uma replicação de dados seria bastante cara. Imaginemos um Site onde seu principal business esteja vinculado ao conteúdo gerado pelos próprios usuários, e que o upload de algum arquivo foi feito para um dos "n" real servers. No momento que o usuário vai acessar o conteúdo gerado, o director o direciona para um real server onde o conteúdo ainda não foi replicado. O usuário poderá achar que sua tentativa falhou e tentará novamente. Com isso, pode-se entrar em questões de credibilidade, que gera prejuizos financeiros e mais uma equipe para tratar duplicações na aplicação, etc.

A idéia é proporcionar servidores com conteúdos sincronizados, onde qualquer modificação será apresentada em real-time. Logo qualquer modificação terá que ser feita no servidor NFS. Obviamente, encontramos mais um serviço para entrar no esquema de Load Balancing Cluster com Failover.

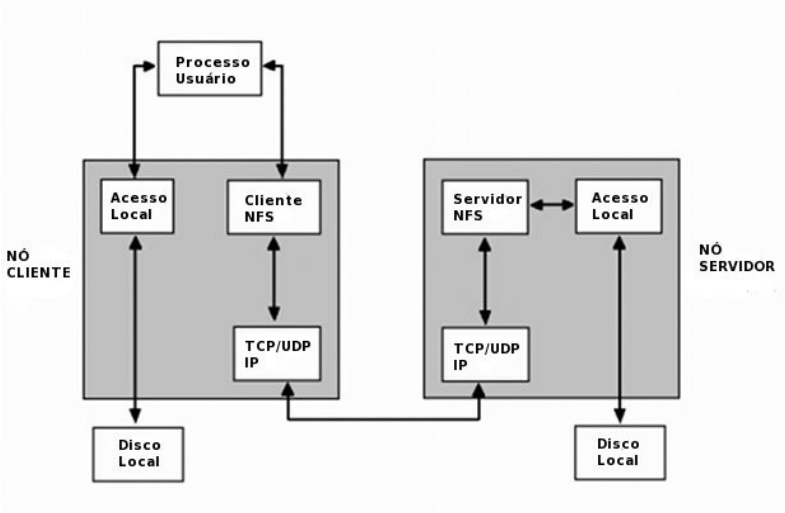


Figura 9 - NFS

4 **OTIMIZAÇÂO DE AMBIENTES ALTAMENTE DISPONÍVEIS**

4.1 REDUZINDO RESPONSE DATA LOAD COM ROTEAMENTO DIRETO

4.1.1 **Necessidade**

Quase sempre um HTTP Request tem um tamanho bastante reduzido, pois apenas o cabeçalho trafega entre o cliente e o servidor. Já o HTTP Response, é muito maior. É nele que são enviadas as imagens, HTML, flash, entre outros. Ao se analisar a técnica LVS-NAT, percebe-se que todo tráfego de entrada e saída passa através do director. Logo a carga de processamento, em sua maior parte, é gerada pelo Response, tornando o director um gargalo na estrutura.

Para resolver este problema o director deveria, apenas, balancear a carga das requisições enquanto os real servers deveriam responder diretamente ao nó requisitante. Porém, este modelo quebraria o funcionamento básico do TCP/IP, onde necessariamente o endereço destino da requisição tem que ser a origem da resposta.

4.1.2 **Roteamento Direto**

Neste modelo, o IP virtual seria compartilhado pelo real servers e o load balancer. Este último, receberia os pacotes de requisição e rotearia diretamente ao real server escolhido. Todos os servidores por trás do director, devem possuir um alias do IP virtual configurado em sua interface, porém não fazendo **arp**. Com isso, estes servidores podem processar o pacote localmente. O director e os real servers devem ter uma de suas interfaces ligada fisicamente a um switch.

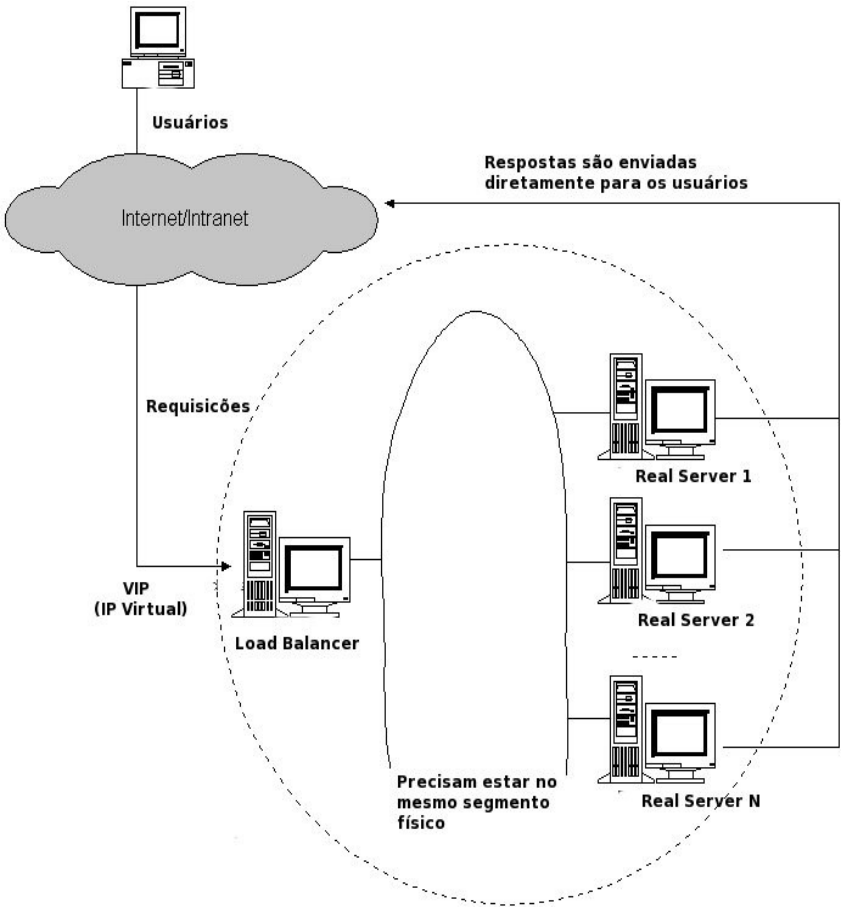


Figura 10 – LVS com Roteamento Direto

Quando um usuário acessa o serviço virtual servido pelo cluster, o pacote é destinado ao endereço VIP. O load balancer examina o endereço de destino e porta do pacote. Se coincidem com um serviço, um real server é escolhido pelo cluster, através de um algoritmo de agendamento, e a conexão é adicionada em uma tabela que armazena as conexões. Então, o load balancer direciona o pacote referido ao real server escolhido. Quando um pacote entrante por esta conexão e o servidor escolhido são encontrados na tabela, ele é novamente roteado diretamente ao servidor. Quando este recebe o pacote direcionado, o servidor verifica se o destino está de acordo com o alias da interface, a requisição é processada e o resultado é retornado diretamente para o usuário final. Após o término ou timeout da conexão, a entrada gravada na tabela é removida.

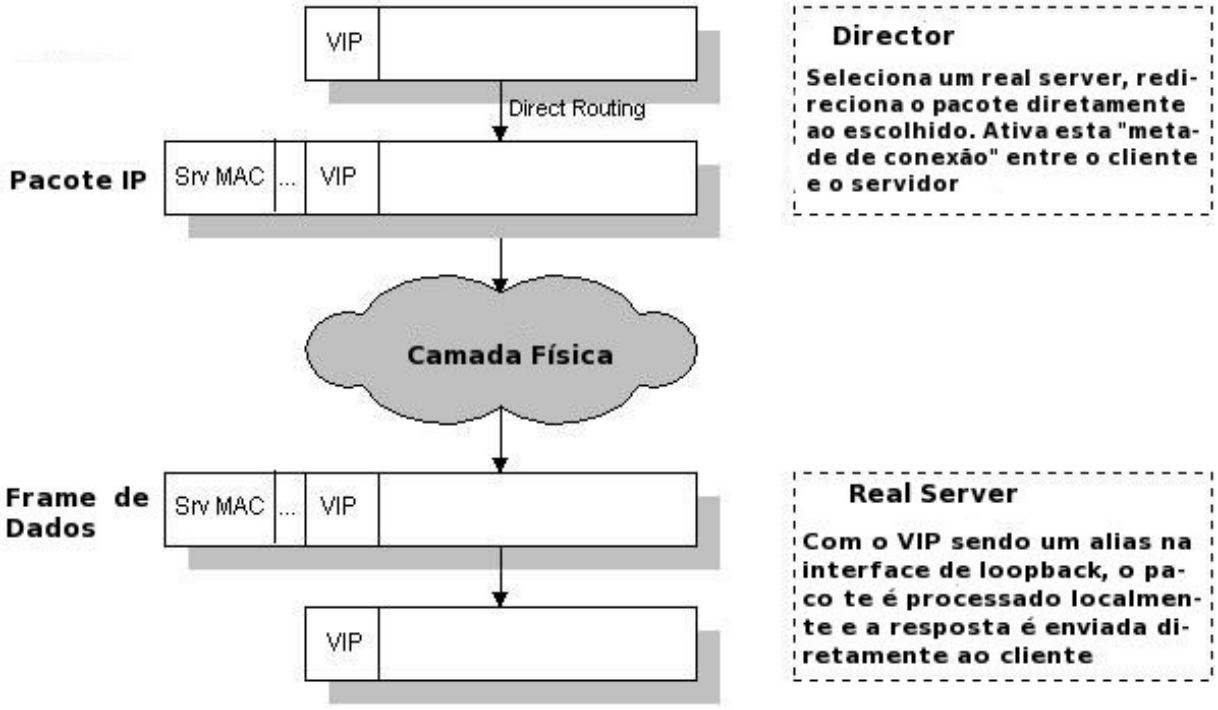


Figura 11 – Roteamento Direto

O load balancer simplesmente modifica o endereço MAC do frame de dados, que é então retransmitido para o servidor escolhido dentro da LAN. Por isso todas as máquinas envolvidas no cluster devem estar diretamente conectadas, sem nenhuma interrupção de segmento dentro da LAN. A dificuldade está em: “Os real servers não podem fazer ARP do VIP, somente o load balancer”.

4.1.3 **Problema com ARP**

Se os real servers responderem o ARP request do VIP, irá começar um corrida sem vencedor. Se algum real server responder a requisição do VIP ocorreria uma falha na integridade da rede, pois o endereço de destino da requisição não seria o mesmo do endereço de origem da resposta. Com isso o cluster pararia de funcionar. Para evitar este problema devemos ter certeza que apenas o director responde ARP para o VIP.

Nas versões 2.0.x e 2.2.x do kernel do Linux e nos sistemas Sun (Solaris e SunOS), qualquer alias criado na interface de loopback não responde ARP. Logo não

teríamos problema em gerar o VIP como alias de uma interface. Já as versões 2.4.x e 2.6.x do kernel do Linux possuem flags built-in para esta funcionalidade:

```
# Habilita a funcionalidade de esconder:
echo 1 > /proc/sys/net/ipv4/conf/all/hidden
# Esconde todos os endereços da interface settada
echo 1 > /proc/sys/net/ipv4/conf/<interface_name>/hidden
```

4.2 CHECAGEM DE REAL SERVERS PELA CAMADA DE APLICAÇÃO

4.2.1 **Necessidade**

Um real server será removido de um cluster em caso de falha ou manutenção de seu serviço. Normalmente esta remoção é feita de forma não amistosa, onde a conexão é interrompida pelo load balancer de forma abrupta quando o serviço não é mais escutado na porta previamente especificada.

Na prática, em um intervalo de tempo configurado, o load balancer verifica se os real servers por trás dele estão escutando em uma determinada porta (configurável). Não conseguindo se conectar, esta máquina é removida do pool de servidores de um serviço específico.

Imaginemos fazer uma manutenção ou simular uma falha com um real server específico, sem que este volte para o pool que está respondendo aos clientes e se quiséssemos simular a interação do mesmo com um banco de dados ou outra aplicação no ambiente de produção ?

Vemos a necessidade de operações de inserção e retirada de real servers do pool formador do cluster de forma rápida e transparente.

4.2.2 **Conceito**

Nosso LVS deve ser capaz de analisar a integridade de uma determinada url, oferecida pelo serviço, afim de tornar possível o switch entre os estados do servidor de forma manual e automática.

A idéia é termos um hash gerado à partir do conteúdo de uma mesma página localizada em todos os real servers. O LVS faria uma checagem num intervalo de

tempo, onde só se manteriam no pool os servers que tivessem uma validação positiva.

Para esta máquina se manter no cluster ela precisa responder sempre o mesmo conteúdo. Para isso ficar preciso, geraríamos um hash da página que seria checada:

Figura 12 – Página de OK (check.html)

página - /check.html

hash - ec90a42b99ea9a2f5ecbe213ac9eba03

No entanto, esta não seria uma página estática, e sim dinâmica. Utilizando-se do servidor de aplicação, colocaríamos uma página dinâmica em cada real server, que responderia de forma diferenciada à partir do momento que um parâmetro específico lhe fosse passado. Como por exemplo o arquivo abaixo:

Quadro 8 (Parte 1) – Conteúdo do arquivo check.jsp

```
<%@ page  contentType="text/html; charset=iso-8859-1"
import="java.sql.*,java.util.*,java.net.*"%>
<%! private static boolean forcedown = false; %>
<% // Checagem de Segurança. Somente rede interna acessa a página %>
String ip = request.getRemoteAddr();
boolean validIP = ip.startsWith("192.168.0.");
```

Quadro 8 (Parte 2) – Conteúdo do arquivo check.jsp

```
if(!validIP) {
        out.println("Permissão Negada");
        return; }
// Passando o parâmentro forcedown=1 tiramos do pool
String forcedown_param = request.getParameter("forcedown");
if(forcedown_param != null) forcedown = forcedown_param.equals("1");
if(forcedown) {
        out.println("Force Down.  Você pode reativa-la no pool passando
o parâmetro forcedown=0 nesta URL");
        return; } %>
Página OK
```

Com uma simples url passada, o seu conteúdo é modificado:

Passando o parâmetro forcedown com o valor 1,

http://server/check.jsp?forcedown=1

O servidor não responderia mais a página com o conteúdo “Página OK”, sendo removida do cluster. E para retornar ao pool sem qualquer restart de serviço (on-the-fly), basta passar o parâmetro com o valor 0 (zero):

http://server/check.jsp?forcedown=0

Atente-se à restrição de segurança apresentada, pois apenas de dentro da rede do cluster (192.168.0.0) é que esta url poderá ser acessada., para não ter problemas com clientes mal-intencionados tentando derrubar o cluster.

4.2.3 **Ferramentas**

Uma excelente ferramenta é o Keepalived (www.keepalived.org). Ele possui um framework robusto de checagem de disponibilidade. Uma das implementações de checagem é do HTTP GET, onde é feita uma checagem remota de integridade do conteúdo de um servidor HTTP.

Trabalhando em layer 5, ele proporciona um GET HTTP em uma url específica. O resultado do GET é somado usando o algoritmo de MD5. Se essa soma não coincidir com o valor esperado, o teste é invalidado e o servidor é removido do pool de servidores. Esse módulo implementa uma checagem multi-URL GET em um mesmo serviço. Essa funcionalidade é muito útil se está sendo usado

um servidor com mais de uma aplicação. O MD5 digest é gerado através do utilitário genhash (incluso no pacote Keepalived).

Quadro 9 - Exemplo de definições da configuração do KeepAlived

```
virtual_server 192.168.200.10 80 {
delay_loop 30
lb_algo wrr
lb_kind NAT
persistence_timeout 50
protocol TCP
sorry_server 192.168.0.100 80

real_server 192.168.0.2 80 {
      weight 2
      HTTP_GET {
            url {
                  path /check.jsp
                  digest ec90a42b99ea9a2f5ecbe213ac9eba03
                  }
             connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
      }
}

real_server 192.168.0.2 80 {
      weight 2
      HTTP_GET {
            url {
                  path /check.jsp
                  digest ec90a42b99ea9a2f5ecbe213ac9eba03
                  }
             connect_timeout 3
            nb_get_retry 3
            delay_before_retry 2
      }
}
```

4.3 DATA CLUSTER COM NFS, RSYNC E VRRP

4.3.1 **Necessidade**

Vimos anteriormente a necessidade da existência de um cluster de dados que seja escalável e síncrono. Em uma empresa com alto índice de manutenção de conteúdo alocado em seu filesystem, juntamento com um pool de servidores tão grande quanto, nos deparamos com a alta criticidade existente no modelo que será adotado.

Todos os servidores web poderiam montar um único servidor NFS resolvendo, aparentemente, os problemas. Porém, este repositório único de dados passa a ser um ponto de falha extremamente alto.

Começam as questões: “O que acontece se o servidor NFS cair ?”, “Os dados alterados durante seu downtime se perderão?”, etc.

Logo tem-se dois pontos principais: **Disponibilidade** e **Integridade**. Será apresentado, agora, como fazer o data cluster tornar-se altamente disponível.

4.3.2 **Disponibilidade**

Para mantermos a disponibilidade, teríamos que adotar a idéia do VIP (Virtual IP). Todas as máquinas estariam montadas em um único IP *Master* via NFS e este estaria conectado via VRRP com outras maquinas-repositório (*Slaves*). Todos os servidores que estivessem em modo *Slave* também teriam o serviço de NFS iniciado.

Todas as máquinas com NFS teriam dois diretórios encontrados na mesma árvore no filesystem. Seria o diretório onde os dados seriam montados e o diretório onde os dados seriam exportados (NFS). Podemos pegar como exemplo o seguinte padrão:

- Ponto de montagem: /mnt/nfs/shared
- Diretório de exportação: /var/exports/nfs/shared

Até mesmo a *Master* montará no VIP. O diretório de exportação da *Master* será sempre o mais atualizado e de tempos em tempos ele sincronizará via rsync este diretório para as outras máquinas (*Slaves*).

Caso aconteça algum problema com o repositório principal, outro publicará seu IP (VIP), já tendo o serviço de NFS iniciado e os diretórios prontos e sincronizados. Ficará transparente para as máquinas que montam.

Quadro 10 – Exemplo de Configuração VIP

```
vrrp_instance VI_1 {
      state MASTER
      interface eth0
      virtual_router_id 51
      priority 150
      advert_int 1
      authentication {
            auth_type AH
            auth_pass k@l!ve1
            }
      virtual_ipaddress {
            192.168.0.1
      }
}
```

4.3.3 **Integridade**

Um problema surge quando um *Slave* assume. Todos os dados gravados no *Master* durante a janela de tempo existente entre o sincronismo de dados do diretório de exportação, ficam perdidos. Isso ocorrerá, pois quando o antigo *Master* voltar, o atual *Master* fará um sincronismo, onde todos os dados que estiverem no antigo e não estiverem no novo serão deletados (sincronismo diferencial).

Para isso deve-se que fazer um esquema diferenciado neste caso. Sempre que uma máquina muda de estado (*Slave/Master*), o KeepAlived permite a excecução de comandos e/ou scripts. Logo, quando uma máquina tornar-se *Master* será feito um sync incremental de duas vias com todas as máquinas do pool. Com isso, todas as máquinas estarão sincronizadas com a *Master*.

Quadro 11 – Exemplo de execução de script em mudança de estado

```
vrrp_instance VI_1 {
      state MASTER
      interface eth0
      virtual_router_id 51
      priority 150
      advert_int 1

      notify_master /path_to_script/script_master.sh
      notify_backup /path_to_script/script_backup.sh

      authentication {
            auth_type AH
            auth_pass k@l!ve1
            }
      virtual_ipaddress {
            192.168.0.1
      }
```

Quando uma máquina que era principal voltar ao pool, provavelmente ela terá dados únicos que devem ser compartilhados com as demais. Tem-se duas opções: sempre que uma máquina voltar ela sempre estará em estado *Master* ou existirá um script de sync incremental de duas vias no seu startup. A primeira opção é perigosa pois talvez não se saiba o porquê da falha da mesma, podendo ocorrer novamente.

5 **MONITORAMENTO**

5.1 PLANEJAMENTO

Quando falamos em planejar um crescimento escalável de um web site, encontramos um grande problema: “Quão rápido será o crescimento de seu tráfego e utilização de seus recursos ?”

A melhor forma de gerar estimativas é através de informações comportamentais por espaço de tempo. O monitoramento do consumo dos recursos envolvidos beneficia de duas formas: uma seria a premissa básica do monitoramento, que é ter uma resposta rápida a um determinado problema, e outra seria a questão estatística que se tem ao longo do tempo.

Logo, é de extrema importância que haja um controle em diversos níveis para maior relevância dos dados colhidos. Tem-se os seguintes recursos de sistema para verificarmos:

- *Consumo de CPU* : Na sua forma mais simples, monitorar a energia da CPU. Consiste em determinar se a utilização da CPU atinge, em algum momento, 100%. Se a utilização da CPU traz algo abaixo de 100%, não importa o que o sistema está fazendo, há energia de processamento disponível para mais carga de trabalho. Entretanto, é raro um sistema que não atinge 100% de utilização da CPU em pelo menos parte do tempo. Neste ponto é importante examinar os dados de utilização mais detalhadamente. Ao fazer isso, é possível começar a determinar onde a maioria da sua energia de processamento está sendo consumida.
- *Largura de Banda* : Monitorar a largura de banda é mais difícil que monitorar outros recursos aqui descritos. A razão disso deve-se ao fato que as estatísticas de desempenho geralmente baseiam-se nos dispositivos, enquanto a maioria dos lugares onde a largura de banda é importante são os canais que conectam dispositivos. Nestes casos, onde mais de um dispositivo compartilha um canal em comum, deve-se observar estatísticas razoáveis para cada dispositivo, mas a carga agregada imposta por estes dispositivos no canal seria bem maior.

- *Memória* : Se existe alguma área onde podemos encontrar uma riqueza de estatísticas de desempenho, essa área é o monitoramento da utilização da memória. Devido à complexidade inerente dos sistemas operacionais com memória virtual paginada por demanda, as estatísticas de utilização da memória são muitas e variadas. É aqui que reside a maioria do trabalho de análise da integração máquina-aplicação.
- *Armazenamento* : O monitoramento do armazenamento geralmente ocorre em dois níveis diferentes: monitoramento de espaço suficiente em disco e monitoramento de problemas de desempenho relacionados ao armazenamento.

É possível ter problemas sérios em uma área e nenhum problema em outra. É possível fazer com que um drive de disco esgote seu espaço sem causar nenhum tipo de problema relacionado ao desempenho. Da mesma forma, é possível ter um drive de disco com 99% de espaço livre com seus limites de desempemnho sendo forçados.

Entretanto, é mais provável que o sistema mediano experimente vários graus de falta de recursos em ambas as áreas. Por causa disso, também é provável que, até certo ponto, os problemas de uma área impactem na outra. Frequentemente, esse tipo de interação toma a forma de desempenho I/O descendente, conforme o drive de disco se aproxima de 0% de espaço livre. Não longe disso, nos casos de cargas I/O extremas, pode ser possível diminuir I/O para um nível no qual as aplicações não mais rodam apropriadamente.

5.2 RRDTool

RRDTool é um sistema de base de dados *round robin* criado por Tobias Oetikersob licença GNU GPL. Foi desenvolvido para armazenar séries de dados numéricos sobre o estado de redes de computadores, porém pode ser empregado no armazenamento de qualquer outra série de dados como temperatura, uso de CPU, etc. RRD é um modo abreviado de se referir a **R**ound **R**obin **D**atabase (base de dados *round-robin*).

A base de dados gerada possui um tamanho máximo o qual uma vez atingido não é ultrapassado. Os dados numéricos armazenados são consolidados conforme

a configuração fornecida, de modo que a resolução deles seja reduzida de acordo com o tempo que eles estão armazenados. Neste processo, apenas as médias dos valores antigos são armazenados.

O RRDTool também pode produzir gráficos que permitem ter uma idéia visual dos dados armazenados, os quais podem ser utilizados ou exibidos por outros sistemas.

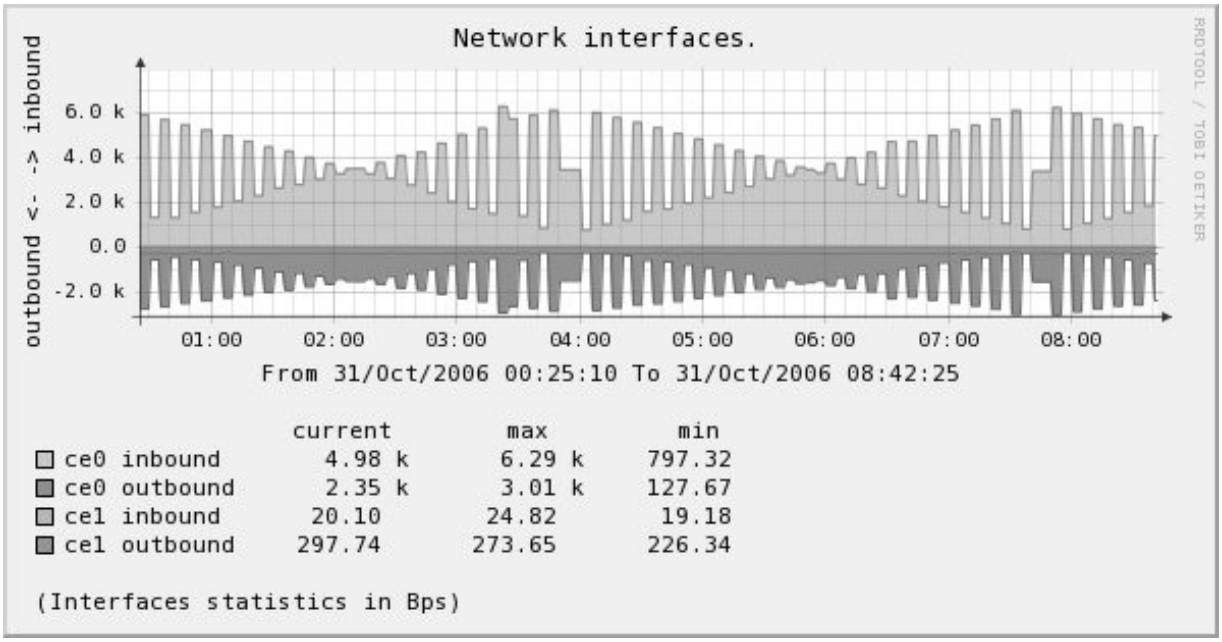


Figura 13 – Exemplo de RRDTool

5.3 CACTI

Cacti é uma ferramenta que recolhe e exibe informações sobre o estado de uma rede de computadores através de gráficos. Foi desenvolvido para ser flexível de modo a se adaptar facilmente a diversas necessidades, bem como ser robusto e fácil de usar. Monitora o estado de elementos de rede e programas, bem como largura de banda utilizada e uso de CPU.

Trata-se de uma interface e uma infra-estrutura para o RRDTool. As informações são repassadas para a ferramenta através de scripts ou outros programas escolhidos pelo usuário, que, por sua vez, devem se encarregar de obter os dados. Pode-se utilizar também o protocolo SNMP para consultar informações em elementos de redes e/ou programas que suportam tal protocolo.

Sua arquitetura prevê a possibilidade de expansão através de plugins que adicionam novas funcionalidades.

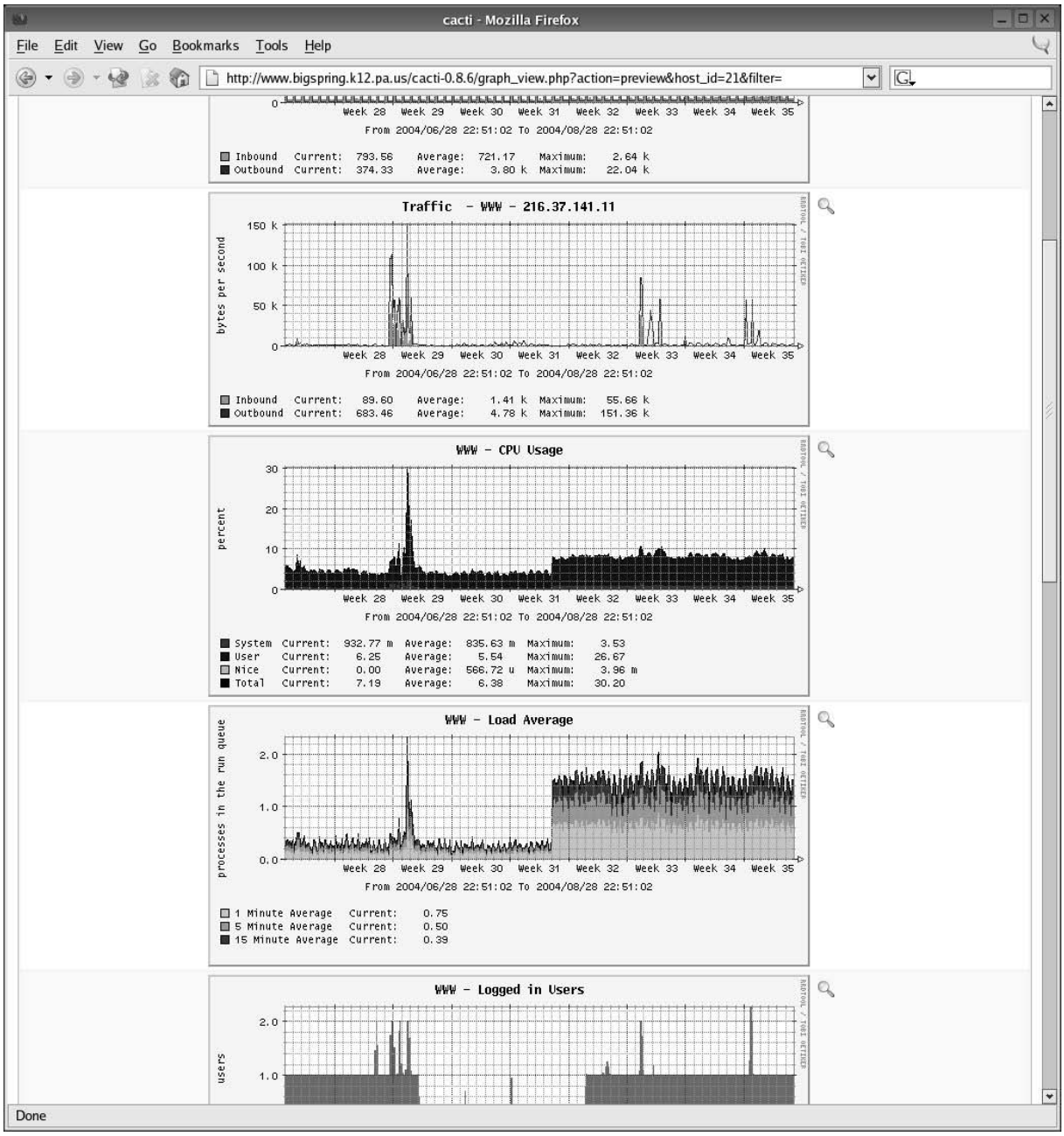


Figura 14 – Cacti

6 **CONCLUSÃO**

Nas empresas de internet a análise do crescimento de tráfego é algo bastante complicado. Diversas variáveis são levantas pelos departamentos de concepção de produtos, que muitas vezes não estão alinhados com a equipe responsável pela manutenção dos servidores web. Uma promoção bem-sucedida, uma ferramenta nova oferecida ao cliente, alguma publicidade aliada ao marketing viral, pode fazer com que o crescimento de tráfego e carga cresça vertiginosamente. O tempo de resposta a esses problemas está diferenciando a qualidade do serviço de diversas empresas.

Além de um bom planejamento, é necessário estar num ambiente onde as soluções tenham espaço com um custo bastante reduzido. Hoje existem diversas ferramentas open-source e a mão-de-obra está bastante qualificada, porém ainda é comum utilizar soluções antigas para problemas extremamente novos. A solução pode estar em resolver problemas de curto prazo com soluções de longo prazo, pois o curto e o longo estão muito próximos na grande rede.

REFERÊNCIAS

Kopper, Karl . **The Linux Enterprise Cluster -** Build a Highly Available Cluster with Commodity Hardware and Free Software, Mai/2005. 464p.

Marcus, Evan; Stern, Hal . **Blueprints for High Availability : Designing Resilient Distributed Systems** . Hoboken, NJ, U.S.A., 2000 . 344p.

Weygant, Peter. **Clusters for High Availability** . 2nd Edition, 2001. 336p.

Linuxvirtualserver.org . **LVS Documents Collection** . Disponível em http://www.linuxvirtualserver.org/Documents.html . Acesso em ago/2007.

Keepalived.org . **HealthChecking for LVS & High-Availability .** Disponível em http://www.keepalived.org . Acesso em ago/2007.

Wensong. **The Keepalived Solution**. Disponível em http://www.linuxvirtualserver.org/docs/ha/keepalived.html . Acesso em set/2007.

Cassen, Alexandre. **Keepalived for LVS - User Guide**. Disponível em http://www.keepalived.org/pdf/UserGuide.pdf . Acesso em set/2007.

Wikipedia . **CARP** . Disponível em http://en.wikipedia.org/wiki/Common_Address_Redundancy_Protocol . Acesso em set/2007.

OpenBSD.org . **Firewall Redundancy with CARP and pfsync** . Disponível em http://www.openbsd.org/faq/pf/carp.html . Acesso em set/2007.

IETF.org . **Virtual Router Redundancy Protocol** . Disponível em http://www.ietf.org/html.charters/vrrp-charter.html . Acessado em set/2007.

IETF.org . **RFC - Virtual Router Redundancy Protocol** . Disponível em http://www.ietf.org/rfc/rfc2338.txt . Acessado em set/2007.

Wikipedia . **VRRP** . Disponível em http://en.wikipedia.org/wiki/Virtual_Router_Redundancy_Protocol . Acesso em set/2007.

Wikipedia . **High Availability** . Disponível em http://en.wikipedia.org/wiki/High-availability_cluster . Acesso em set/2007.

Sourceforge.net . **Linux NFS Overview, FAQ and HOWTO Documents**. Disponível em http://nfs.sourceforge.net/ . Acessado em out/2007

Wikipedia . **NFS** . Disponível em http://en.wikipedia.org/wiki/Network_File_System (protocol) . Acesso em set/2007.

Apache.org . **The Apache HTTP Server Project** . Disponível em http://httpd.apache.org/ . Acesso em ago/2007.

Wikipedia . **Web Server** . Disponível em http://en.wikipedia.org/wiki/Web_Server . Acesso em ago/2007.

Samba.edu . **RSync Documentation**. Disponível em http://samba.anu.edu.au/rsync/documentation.html . Acessado em out/2007.

Wikipedia . **RSync .** Disponível em http://en.wikipedia.org/wiki/RSync . Acesso em out/2007.

Oetiker.ch. **About RDDTool** . Disponível em http://oss.oetiker.ch/rrdtool/ . Acesso em nov/2007.

Wikipedia . **RRDTool .** Disponível em http://pt.wikipedia.org/wiki/RRDTool . Acesso em nov/2007.

Cacti.net . **About Cacti** . Disponível em http://cacti.net/ . Acesso em nov/2007.

Wikipedia . **Cacti .** Disponível em http://pt.wikipedia.org/wiki/Cacti . Acesso em nov/2007.